Implementació de la cerca d'un subgraf comú maximal a dos grafs mitjançant colles

Universitat de Lleida
Escola Politècnica Superior

TFC Enginyeria Informàtica

9 de juny de 2011

Als meus pares, Plácido López i Pepita Masip,

per haver-me educat i ensenyat de la manera que ho han fet,

per haver-me empès a estudiar, a tenir valors i marcar-me fites en aquesta vida i, sobretot,

perquè sense ells no hauria estat, mai, res de res.

Agraïments

A en Joan Gimbert, Josep Conde i Nacho López,
per haver-me ensenyat, guiat i animat en el transcurs del TFC.

A la meva família, xicota, amics i companys,
per haver tingut i tenir tanta paciència amb mi
(suportar-me cada dia, sí que és un problema de cost exponencial).

Als meus pares,
en aquest cas, perquè m'han donat l'oportunitat de ser català, i de Lleida.

Índex

Llista d'algorismes

Índex de figures

Índex de taules

Introducció

La memòria que ací es presenta s'emmarca dins de l'àrea de teoria de grafs. En concret es treballa la implementació d'un algorisme per trobar el subgraf comú maximal (SCM) de dos grafs mitjançant la cerca de colles maximals (CM). Aquest projecte s'ha desenvolupat a la facultat d'informàtica de la Universitat de Lleida (UdL).

Com a punt de partida, s'han estudiat diversos articles que tracten de la distància entre grafs, essent els principals: "Una mètrica de distància de grafs basada en el SCM", de Horst Bunke i Kim Shearer [1], "Una comparativa d'algorismes per trobar el SCM en grafs connexos aleatoris" de Horst Bunke, Pasquale Foggia, Corrado Guidobaldi, Carlo Sansone i Mario Vento [2] i "El problema del subgraf comú maximal" de Faisal N. Abu-Khzam, Nagiza F. Samatova, Mohamad A. Rizk i Michael A. Langston [3].

L'aportació principal del projecte consisteix en, donats dos grafs qualssevol, trobar el seu graf associat per tal de poder cercar la seva colla maximal (CM). I així, utilitzant funcions existents en el llenguatge de programació, poder trobar el seu subgraf comú maximal (SCM), necessari per calcular la distància entre grafs i així determinar quan d'isomorfs són.

Recordem que la identificació de grafs isomorfs és un problema clàssic en la teoria de grafs, no resolt. En aquest projecte s'ha abordat des d'una vessant reduïda a partir de l'estudi de les colles maximals. Problema que ja per si sol, és prou difícil.

Els resultats obtinguts en els càlculs han deixat palesa la dificultat de l'esmentat problema i dels costos temporals que les funcions emprades comporten, essent les darreres d'ordre exponencial.

La memòria està organitzada en sis parts. En el Capítol 1 es donen els fonaments matemàtics bàsics i necessaris per tal de poder entendre els fonaments teòrics de l'esmentada. En el Capítol 2 es mostren els principals articles que han servit de base per l'estudi del problema del subgraf comú maximal. En el Capítol 3 es fa una breu explicació de les funcions de la llibreria NetworkX que o bé estan relacionades amb colles o bé s'han utilitzat per la implementació dels programes. En el Capítol 4 es mostra el codi per tal de generar el graf associat, l'obtenció de la colla maximal i el càlcul de la distància necessaries per la consecució de l'objectiu final. En el Capítol 5 es mostren les dades calculades amb els programes generats i explicats en el capítol anterior. I, finalment, es completa la memòria amb el Capítol 6, que correspon al resum, conclusions i treball futur.

Capítol 1

Fonaments matemàtics

En aquest primer capítol veurem els conceptes bàsics necessaris perquè el lector pugui entendre sense massa dificultat els fonaments matemàtics que fan referència a la teoria de grafs [4] que s'amaguen devall dels algorismes que s'han realitzat en el treball que tenen a les mans.

1.1 Grafs: conceptes bàsics

1.1.1 Definició de graf

Un *graf* $G = (V, A)$ està format per un conjunt d'elements finit i no buit V i per un conjunt A de parells no ordenats d'elements diferents de V. Veure Fig. 1.1.

Als elements de V els anomenarem *vèrtexs* i als elements d'A *arestes*. Si $a = \{u, v\}$ és una aresta, direm que u i v són vèrtexs adjacents, i ho escriurem com $u \sim v$, i també direm que l'aresta a és incident amb els vèrtexs u i v. Normalment, per abreujar la notació escriurem $a = uv$ enlloc de $a = \{u, v\}$.

1.1.2 Ordre i mida d'un graf

L'*ordre* d'un graf $G = (V, A)$ és el nombre de vèrtexs de G, és a dir, el cardinal de V, que denotarem amb $|V|$. La *mida* de G és el nombre d'arestes de G, que denotarem amb $|A|$.

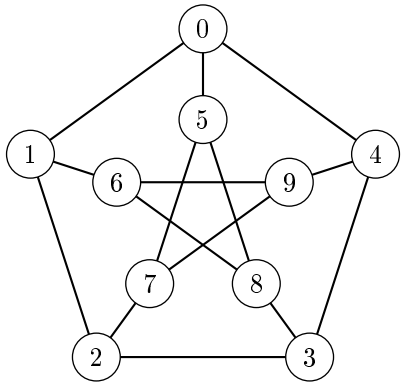

Figura 1.1: Graf de Petersen

1.1.3 Grau d'un vèrtex i graf d-regular

El *grau* d'un vèrtex v en un graf $G = (V, A)$, que denotarem per $g(v)$, és el nombre d'arestes de G incidents amb v.

Un graf $G = (V, A)$ és $d - regular$ si tots els vèrtexs de G tenen grau d. En el cas de l'exemple de la Figura 1.2 el graf G_1 és 1-regular i el graf G_2 és 2-regular.

1.1.4 Camí i distància

Un *camí* de longitud l és un graf $P = (V, E)$ de la forma $V = \{x_0, x_1, \ldots, x_l,\ \ x_i \neq x_j,\ \ 1 \leq i < j \leq l\}$, $E(P) = \{x_0x_1, x_1x_2, \ldots, x_{l-1}x_l\}$. Habitualment, diem que un camí $P = \{x_0, \ldots, x_l\}$ és un $x_0 - x_l$ camí.

La *distància* entre dos vèrtexs u i v de G, que denotarem per $d(u, v)$, és la mínima de les longituds dels $u - v$ camins. I aquesta distància és una *mètrica* en el conjunt de vèrtexs V, ja que satisfà les propietats següents:

1. $d(u, v) \geq 0$ i $d(u, v) = 0$ si i només si $u = v$;
2. $d(u, v) = d(v, u)$;
3. $d(u, v) \leq d(u, w) + d(w, v)$ (desigualtat triangular).

1.1.5 Excentricitat, radi i diàmetre

L'*excentricitat* d'un vèrtex v de G, denotada per $e(v)$, és la distància més gran entre v i els altres vèrtexs de G. El *radi* de G, denotat per $r(G)$ és la més petita de les excentricitats. I, el *diàmetre* de G, denotat per $D(G)$, és la màxima de les distàncies entre els vèrtexs de G, és a dir,

$$D(G) = màx.\ \{d(u, v) \mid u, v \in V\}$$

1.1.6 Graf complet i graf complementari

Sigui $G = (V, A)$ un graf d'ordre n, direm que G és un *graf complet*, que denotarem per K_n, si cada vèrtex és adjacent a tots els vèrtexs restants del graf, és a dir,

$$K_n = (V,\ \{uv \mid \forall u, v \in V,\ u \neq v\})$$

Sigui $G = (V, A)$ un graf d'ordre n, direm que $\bar{G} = (V, \bar{A})$ és el seu graf complementari, i dos vèrtexs diferents $u, v \in V$ són adjacents en $\bar{G}$, si i només si no ho són en G.

1.1.7 Graf associat

Siguin $G_1 = (V_1, A_1)$ i $G_2 = (V_2, A_2)$ dos grafs qualssevol, direm que el *graf associat* de G_1 i G_2, que denotarem per $G_{A(G_1G_2)}$, és el graf $G_{A(G_1G_2)} = (V_A, A_A)$ on el conjunt de vèrtexs serà $V_A = V_1 \times V_2$ i un parell de vèrtexs (u_1, v_1) i (u_2, v_2) són adjacents sempre i quan: $u_1 \neq u_2$ i $v_1 \neq v_2$ i

$$\begin{cases} (u_1, u_2) \in A_1 \quad i \quad (v_1, v_2) \in A_2 \\ \qquad\qquad o\ bé, \\ (u_1, u_2) \notin A_1 \quad i \quad (v_1, v_2) \notin A_2 \end{cases}$$

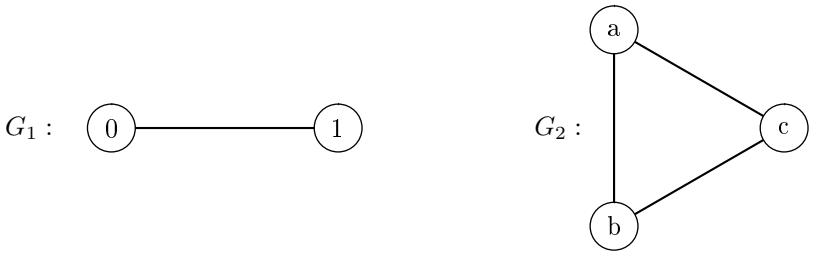

Figura 1.2: Grafs G_1 i G_2

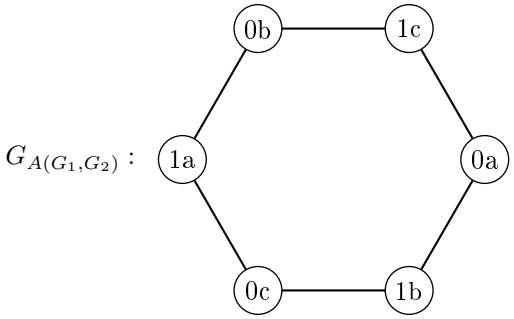

Figura 1.3: Graf associat dels grafs G_1 i G_2

1.1.8 Cota de Moore, graf de Moore i graf radial de Moore

Siguin d el grau màxim i k el diàmetre d'un graf, dos nombres naturals, es defineix la *cota de Moore* del graf, com:

$$M_{d,k} = d\sum_{i=1}^{k}(d-1)^{i-1} = 1 + d + d(d{-}1)^1 + \cdots + d(d{-}1)^{k-1}$$

El valor $M_{d,k}$ és la cota superior del nombre de vèrtexs que pot tenir un graf de grau màxim d i diàmetre k. I, es verifica que $n \leq M_{d,k}$ per tot graf d'ordre n.

Sigui $G = (V, A)$ un graf d'ordre n, direm que és un *graf de Moore* si $n = M_{d,k}$, és a dir, si el graf arriba a la cota de Moore. D'aquests darrers es coneixen els següents:

- L'únic graf de Moore amb $d = 1$ és el K_2.
- De grafs de Moore amb $d = 2$ n'hi ha infinits, són els que corresponen als cicles de longitud $2k + 1$.
- I sol existeixen (o pot ser que existeixin) tres grafs de Moore amb $d \geq 3$. Aquests grafs tenen diàmetre 2 i grau 3, 7 i possiblement 57.

S'anomenen *grafs radials de Moore* als grafs regulars de grau d, radi k, diàmetre $k+1$ i ordre igual a $M_{d,k}$.

1.2 Subgrafs: conceptes bàsics

1.2.1 Definició de subgraf i subgraf generador

Un graf $G' = (V', A')$ és un *subgraf* de $G = (V, A)$ si $V' \subseteq V$ i $A' \subseteq A$. Quan $V' = V$ direm que V' és un *subgraf generador* de G.

1.2.2 Subgraf induït

Sigui $G = (V, A)$ un graf i S un subconjunt de V, el subgraf induït per S, que denotarem per $\langle S \rangle$, és el graf$\langle S \rangle = (S, A')$, on els elements de A' són les arestes de G que uneixen vèrtexs de S, és a dir,

$$A' = \{uv \in A \mid u \in S \; i \; v \in S\}.$$

En cas que $S = V - \{v\}$, el subgraf induït, que es denota per $G - v$, s'obté al suprimir de G el vèrtex v i les arestes incidents amb aquest vèrtexs. Anàlogament, si $W \subseteq V$ el graf $G - W$ denota el graf obtingut al suprimir de G els vèrtexs de W i les arestes incidents amb aquests vèrtexs.

En canvi, la supressió d'una aresta a d'un graf $G = (V, A)$ dóna lloc, per definició, al subgraf $G - a = (V, A - \{a\})$, és a dir, s'elimina l'aresta a però no els vèrtexs incidents amb aquesta aresta.

Si u i v són dos vèrtexs no adjacents d'un graf $G = (V, A)$, definim que:

$$G + uv = (V, A \cup \{uv\}),$$

és a dir, $G + uv$ és el graf que s'obté de G a l'afegir l'aresta uv.

1.2.3 Subgraf comú

Siguin $G_1 = (V_1, A_1)$ i $G_2 = (V_2, A_2)$ dos grafs qualsevols, direm que $G = (V, A)$ és un *subgraf comú* de G_1i G_2, que denotarem per $SC(G_1, G_2)$, si existeix un subgraf isomorf (l'isomorfisme és definit en la subsecció següent) de G a G_1 i de G a G_2.

1.2.4 Subgraf comú maximal (SCM)

Siguin $G_1 = (V_1, A_1)$ i $G_2 = (V_2, A_2)$ dos grafs qualssevol, direm que $G = (V, A)$ és un *subgraf comú maximal* de G_1 i G_2, que denotarem per $SCM(G_1, G_2)$, si no existeix cap altre subgraf comú de G_1i G_2 que sigui d'un ordre major que el de G.

S'ha de tenir en compte que el $SCM\,(G_1,\, G_2)$ no ha d'ésser necessàriament únic pels grafs donats. I, d'acord a la major part de la literatura estudiada, és possible que els esmentats tinguin vèrtexs aïllats.

1.3 Isomorfisme

1.3.1 Definició d'isomorfisme

Direm que dos grafs $G = (V, A)$ i $G' = (V', A')$ són $isomorfs$, i ho denotarem per $G \simeq G'$, si existeix una aplicació bijectiva f de V en V' tal que preserva les adjacències; és a dir,

$$\forall u, v \in V,\, u \sim v \Leftrightarrow f(u) \sim f(v).$$

Podem dir d'una manera intuïtiva, que dos grafs isomorfs només es diferencien per la manera que se n'etiqueten els vèrtexs i, en general, per la seva representació gràfica. Els vèrtexs de dos grafs isomorfs es poden identificar un a un de manera que les adjacències de l'un i de l'altre siguin les mateixes, és a dir, l'estructura dels dos grafs, un cop identificats els vèrtexs corresponents, és idèntica. Veure Fig. 1.4.

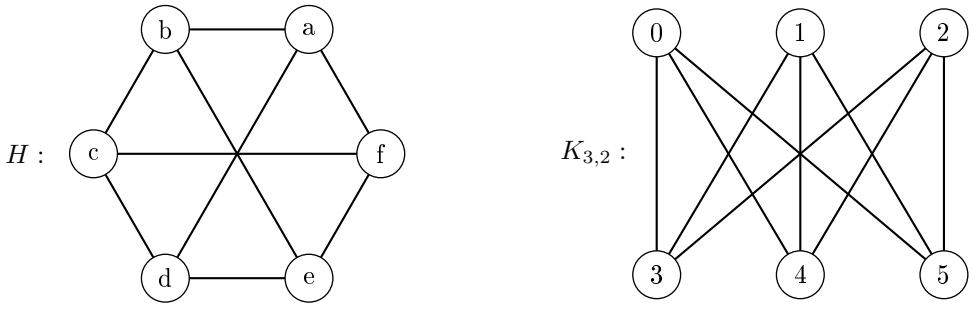

Figura 1.4: Exemple de grafs isomorfs

1.3.2 Isomorfisme d'arestes

Direm que dos grafs $G = (V, A)$ i $G' = (V', A')$ són $aresta - isomorfs$, i ho denotarem per $G \simeq_A G'$, si existeix una aplicació bijectiva f d'A en A' tal que dues arestes a_1i a_2 de G són adjacents si, i només si, les arestes $f(a_1)$ i $f(a_2)$ són adjacents en G'; és a dir,

$$\forall a_1, a_2 \in A,\ a_1 \sim a_2 \Leftrightarrow f(a_1) \sim f(a_2).$$

1.4 Colla (*Clique*)

1.4.1 Definició de colla

La paraula anglesa *clique* significa: 'grup de persones que comparteix uns interessos en comú'. On, per analogia, les persones són els vèrtexs i els interessos que comparteixen les arestes.

En matemàtica discreta un *clique* d'un graf $G = (V, A)$, d'ara en endavant *colla*, es defineix com el subgraf complet induït per V' on $V' \subseteq V$, i es denota per $C(G)$. És a dir,

$$C(G) = (V' : V' \subseteq V,\ A' : \{\forall u, v \in V',\ u \sim v \Leftrightarrow u \neq v\}).$$

1.4.2 Colla maximal (CM)

Sigui $G = (V, A)$ un graf, direm que una colla $C(G)$ és un *colla maximal*, i ho denotarem per $CM(G)$, si no existeix cap altra colla d'ordre més gran. Veure Fig. 1.5.

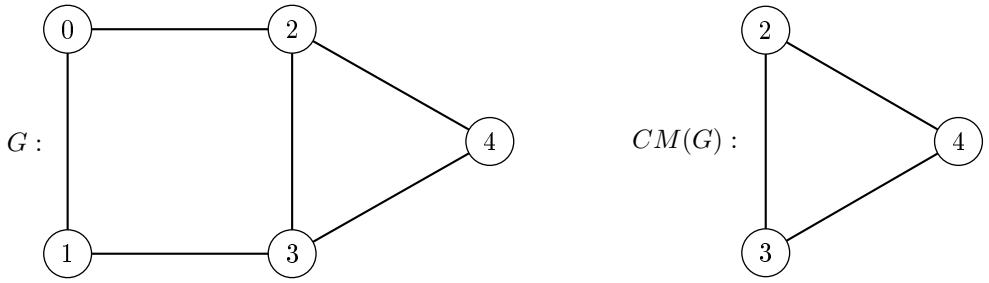

Figura 1.5: Exemple d'un graf G i la CM(G)

1.5 Relació entre la CM del graf associat i el SCM

La relació que hi ha entre la colla maximal del graf associat i el subgaf comú maximal de dos grafs és la seva estructura, és a dir, el conjunt de vèrtexs i arestes que tindran o no en comú.

L'esmentada afirmació es pot deduir de l'observació de la colla maximal del graf associat i de la pròpia definició de les seves adjacències, com veurem a continuació:

En l'anterior apartat s'ha vist que una colla maximal és un subgraf complet induït d'un nombre determinat de vèrtexs. Llavors, quan agafem una aresta qualsevol de la colla maximal del graf associat sabem que aquesta, en els grafs inicials, es tradueix com que hi ha una aresta d'un vèrtex a un altre per ambdós grafs inicials o que l'aresta no hi és per cap dels dos, és a dir, que existeix una no-aresta incident a un parell de vèrtexs del primer graf i una no-aresta que és incident a un parell de vèrtexs del segon graf. I aquesta disjunció fa que es mantingui l'estructura.

És a dir, siguin $G_1 = (V_1, A_1)$ i $G_2 = (V_2, A_2)$ dos grafs qualssevol, si existeix una aresta en el graf associat $G_{A(G_1G_2)} = (V_A, A_A)$ es pot deduir que:

1. Existeix una aresta que és incident a dos vèrtexs del graf G_1 i a dos vèrtexs de G_2, que es dedueix de:

$$(u_1, u_2) \in A_1 \quad i \ (v_1, v_2) \in A_2$$

2. Eisteix una no-aresta que és incident a dos vèrtexs de G_1 i a dos vèrtexs de G_2, que es dedueix de:

$$(u_1, u_2) \notin A_1 \quad i \ (v_1, v_2) \notin A_2$$

Capítol 2

Articles

En aquest capítol es parla d'alguns dels treballs realitzats que tenen per objecte d'estudi el càlcul del subgraf comú maximal i altres conceptes afins com, per exemple, el càlcul de la distància entre grafs.

2.1 Una mètrica de distància de grafs basada en el SCM

2.1.1 Introducció

En aquest article es proposa un nou mètode per calcular la distància entre grafs, que consisteix en trobar el subgraf comú maximal entre dos grafs donats. La principal contribució de l'article és la prova formal que la nova mesura de la distància és una mètrica.

2.1.2 Definicions bàsiques

En l'article es consideraran els vèrtexs i les arestes etiquetats. L_V i L_A denotaran el conjunt finit d'etiquetes de vèrtexs i arestes, respectivament.

Definició 1: Un *graf* és una 4-pla[1] $G = (V, A, \mu, \nu)$, on:

- V és un conjunt finit de vèrtexs, si $V = \emptyset$ s'anomenarà graf buit.
- A és un conjunt d'arestes,
- $\mu : V \to L_V$ és una funció que assigna etiquetes als vèrtexs
- $\nu : A \to L_A$ és una funció que assigna etiquetes al les arestes

Definició 2: Donat un graf $G = (V, A, \mu, \nu)$, un *subgraf* de G és un graf $G = (V_S, A_S, \mu_S, \nu_S)$ tal que:

- $V_S \subseteq V$,
- $A_S = A \cap (V_S \mathrm{x} V_S)$,
- μ_Si ν_S són les restriccions de μ i ν a V_S i A_S, respectivament:

$$\mu_S(v) = \begin{cases} \mu(v) & si\ v \in V_s, \\ sense\ definir & altrament. \end{cases}$$

[1] Sigui n és un nombre natural, aleshores una n-pla (de vegades n-tupla) és una seqüència o llista ordenada de n objectes, i aquests elements es diu que són les seves components.

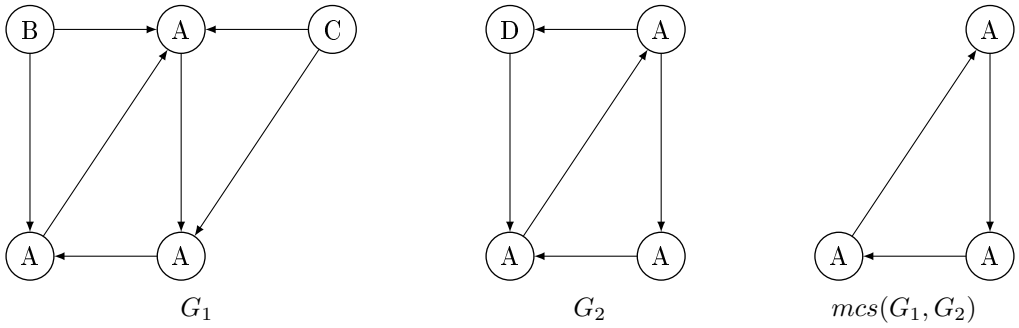

Figura 2.1: Un exemple del càlcul de la distància

$$\nu_S(a) = \begin{cases} \nu(a) & si\ a \in A_s, \\ sense\ definir & altrament. \end{cases}$$

La notació $S \subseteqq G$, s'utilitza per indicar que S és un subgraf de G.

Definició 3: Una funció bijectiva $f : V \to V'$ és un *isomorfisme de graf* des d'un graf $G = (V, A, \mu, \nu)$ a un graf $G' = (V', A', \mu', \nu')$ si:

- $\mu(v) = \mu'(f(v)), \forall v \in V$,
- $\forall a,\ a = (v_1, v_2) \in A,\ \exists\, a' : (f(v_1), f(v_2)) \in A'$ tal que $v(a) = v(a')$ i,
- $\forall a',\ a' = (v'_1, v'_2) \in A',\ \exists\, a : (f^{-1}(v'_1), f^{-1}(v'_2)) \in A$ tal que $v(a') = v(a)$

Definició 4: Una funció injectiva $f : V \to V'$ és un isomorfisme de subgraf des d'un graf G a un graf G' si existeix un subgraf $S \subseteqq G'$ tal que f és un isomorfisme de graf de G a S.
Trobar un isomorfisme de subgraf de G' a G implica trobar un subgraf de G' isomòrfic a tot G.

Definició 5: Siguin G, G_1 i G_2 grafs, G és un *subgraf comú* de G_1 i G_2 si existeix un subgraf isomorf de G a G_1 i de G a G_2.

Definició 6: Un subgraf comú G de G_1 i G_2 és *maximal* si no existeix cap altre subgraf comú G' de G_1 i G_2 que tingui més vèrtexs que G.

El subgraf comú maximal de dos grafs G_1 i G_2, serà denotat per $mcs(G_1, G_2)$. S'ha de tenir en compte que no ha d'ésser necessàriament únic.

2.1.3 Mesura de la distància d'un graf

Definició 7: La *distància* entre dos grafs no buits G_1 i G_2 es defineix com:

$$d(G_1, G_2) = 1 - \frac{|mcs(G_1, G_2)|}{max(|G_1|, |G_2|)}$$

A la Fig. 2.1 es mostra un exemple del càlcul de la distància on $|G_1| = 5$, $|G_2| = 4$ i $|mcs(G_1, G_2)| = 3$. Per tant $d(G_1, G_2) = 0.4$.

Teorema: Per qualsevol graf G_1, G_2 i G_3, les següents propietats es consideren certes:

1. $0 \leq d(G_1, G_2) \leq 1$,
2. $d(G_1, G_2) = 0 \Leftrightarrow G_1 \, i \, G_2$ són isomorfs entre si,
3. $d(G_1, G_2) = d(G_2, G_1)$,
4. $d(G_1, G_3) \leq d(G_1, G_2) + d(G_2, G_3)$.

Demostració: Les propietats 1-3 es dedueixen directament de la definició número 7. En la següent demostració de la desigualtat triangular es distingiran 2 casos:

Cas A:
Els grafs $mcs(G_1, G_2)$ i $mcs(G_2, G_3)$ són disjunts o, dit d'una altra manera, el subgraf comú maximal de $mcs(G_1, G_2)$ i $mcs(G_2, G_3)$ és buit.

Sigui $m_{12} = |mcs(G_1, G_2)|$, $m_{23} = |mcs(G_2, G_3)|$ i $m_{13} = |mcs(G_1, G_3)|$ es considera certa la relació:

$$m_{12} + m_{23} \leq |G_2| \,. \tag{2.1}$$

La quarta propietat del teorema és equivalent a la desigualtat següent:

$$1 - \frac{m_{12}}{max(|G_1|, |G_2|)} + 1 - \frac{m_{23}}{max(|G_2|, |G_3|)} \geq 1 - \frac{m_{13}}{max(|G_1|, |G_3|)} \tag{2.2}$$

La part esquerra de la desigualtat és sempre més gran o igual a 1, que és equivalent a:

$$max(|G_1|, |G_2|) \; max(|G_2|, |G_3|) \geq m_{12} max(|G_2|, |G_3|) + m_{23} max(|G_1|, |G_2|) \tag{2.3}$$

Partirem d'un cas d'anàlisi senzill.
Cas A.1:

$$|G_1| \geq |G_2| \geq |G_3| \,.$$

Ací l'equació (2.3) és equivalent a:

$$|G_1| \cdot |G_2| \geq m_{12} \, |G_2| + m_{23} \, |G_1| \tag{2.4}$$

De l'equació (1) arribem a la conclusió que:

$$|G_1| \, |G_2| \geq m_{12} \, |G_2| + m_{23} \, |G_1| \geq m_{12} \, |G_2| + m_{23} \, |G_1| \,.$$

Cas A.2:

$|G_1| \geq |G_3| \geq |G_2|$. Ací l'equació (2.3) és equivalent a:

$$|G_1| \cdot |G_3| \geq m_{12} |G_3| + m_{23} |G_1| \tag{2.5}$$

De l'equació (2.1) arribem a la conclusió que:

$$|G_1| |G_3| \geq |G_1| |G_2| \geq m_{12} |G_1| + m_{23} |G_1| \geq m_{12} |G_3| + m_{23} |G_1| .$$

Els altres quatre casos restants es faran de manera similar als anteriors:

$$|G_2| \geq |G_1| \geq |G_3| , \quad |G_2| \geq |G_3| \geq |G_1| ,$$
$$|G_3| \geq |G_1| \geq |G_2| , \quad |G_3| \geq |G_2| \geq |G_1| .$$

Cas B:
Ací s'assumeix que el subgraf comú maximal de $mcs(G_1, G_2)$ i $mcs(G_2, G_3)$ no és buit.
Llavors, sigui $m = |mcs(mcs(G_1, G_2), mcs(G_2, G_3))| > 0$. Es dedueix que existeix un SCM de G_1 i G_3 amb una mida més gran o igual a m. A més es dedueix que

$$m_{12} + m_{23} - m \leq |G_2| , \qquad m \leq m_{12}, \qquad m \leq m_{23} \tag{2.6}$$

Anem a demostrar que

$$1 - \frac{m_{12}}{max(|G_1| , |G_2|)} + 1 - \frac{m_{23}}{max(|G_2| , |G_3|)} \geq 1 - \frac{m}{max(|G_1| , |G_3|)} \tag{2.7}$$

que implica la quarta propietat del teorema. Obviament, la desigualtat (2.7) és equivalent a

$$max(|G_1| , |G_2|)max(|G_2| , |G_3|)max(|G_1| , |G_3|) \geq m_{12}max(|G_2| , |G_3|)max(|G_1| , |G_3|)+$$
$$+m_{23}max(|G_1| , |G_2|)max(|G_1| , |G_3|) - m \cdot max(|G_1| , |G_2|)max(|G_2| , |G_3|). \tag{2.8}$$

El seu anàlisi és:
Cas B.1:

$$|G_1| \geq |G_2| \geq |G_3| .$$

Ací l'equació (2.8) és equivalent a

$$|G_1| |G_1| |G_2| \geq m_{12} |G_1| |G_2| + m_{23} |G_1| |G_1| - m |G_1| |G_2|$$

que es pot simplificar com

$$|G_1| |G_2| \geq m_{12} |G_2| + m_{23} |G_1| - m |G_2| = (m_{12} - m) |G_2| + m_{23} |G_1| . \tag{2.9}$$

De l'equació (2.6) es dedueix que

$$|G_1| |G_2| \geq m_{12} |G_1| + m_{23} |G_1| - m |G_1| = (m_{12} - m) |G_1| + m_{23} |G_1|$$

des d'on obtenim (2.9) ja que $m_{12} \geq m$.

Cas B.2:

$$|G_1| \geq |G_3| \geq |G_2| \,.$$

Ací l'equació (2.8) és equivalent a:

$$|G_1|\,|G_1|\,|G_3| \geq m_{12}\,|G_1|\,|G_3| + m_{23}\,|G_1|\,|G_1| - m\,|G_1|\,|G_3|$$

que es pot simplificar com

$$|G_1|\,|G_3| \geq m_{12}\,|G_3| + m_{23}\,|G_1| - m\,|G_3| \,. \tag{2.10}$$

I, anàlogament al cas B.1

$$|G_1|\,|G_3| \geq |G_1|\,|G_2| \geq m_{12}\,|G_1| + m_{23}\,|G_1| - m\,|G_1| \geq$$

$$\geq m_{12}\,|G_3| + m_{23}\,|G_1| - m\,|G_3| \,. \tag{2.11}$$

Per la resta de casos es demostraria de la mateixa manera.

2.1.4 Conclusions

Tal i com hem pogut veure en els apartats anteriors s'ha demostrat que la mesura de distàncies de grafs de la definició 7 és una mètrica.

Els algorismes classics per obtenir el SCM estan basats en la detecció de la colla maximal de Levi (1972) [5] o en el retrocés (*backtracking*) de McGregor (1982) [6].

En un treball recent s'ha demostrat que el càlcul del SCM pot considerar-se un cas especial del càlcul de distàncies entre grafs sobre una funció de costos particular (Bunke, 1997 [7]). I, una conseqüència immediata és que qualsevol algorisme per al càlcul de la distàncies entre grafs es pot utilitzar per calcular el SCM si s'executa sota la fució donada per Bunke (1997). I, per tant, deixa obertes possibilitat adicionals per el càlcul de les mesures de distàncies proposades en aquest treball.

2.2 Una comparativa d'algorismes per trobar el SCM en grafs connexos aleatoris

2.2.1 Introducció

En aquest article es descriuen dos algorismes per trobar el SCM i el seu rendiment, en una base de dades [6] que conté parells de grafs connectats aleatòriament, essent el SCM d'almenys un parell de nodes.

Tot i que hi ha un nombre significant d'algorismes per a detectar el SCM, fins al moment, no s'havia esmerçat cap esforç per calcular el seu rendiment. Per tant, el comportament dels esmentats algormismes no és massa clar per a mides de grafs que s'han d'ajustar als canvis que hi ha d'una aplicació a una altra. La manca d'una base de dades de grafs comuna complica bastant la tasca de comparar el rendiment dels diferents algorismes per trobar el SCM.

En els següents apartats es presenten dos algorismes que segueixen diferents principis per a la cerca assenyalada. El primer trobarà el SCM fent la cerca de tots els subgrafs comuns. I, el segon, el trobarà calculant el CM un cop construït el graf associat.

2.2.2 Un algorisme de cerca d'estats d'espai per detectar el SCM

Per l'algorisme proposat en aquesta secció es calcularà el subgraf maximal que conté el màxim nombre de vèrtexs. I deriva de l'algrisme descrit per McGregor [6]. A continuació podem veure un esbós:

Algorithm 2.1 Esbós de la cerca de l'estat d'espai del SCM

```
Procedure MCS (s,n1,n2)
Begin
    if (NextPair (n1,n2)) then
    Begin
        if (IsFeasiblePair (n1,n2)) then
        AddPair (n1,n2);
        CloneState (s,s');
        while(s' is not a leaf of the search tree)
        Begin
            MCS (s',n1,n2);
            Backtrack (s');
        end
        Delete (s');
    end
end procedure
```

On, cada estat s representa un subgraf comú dels dos grafs en construcció i és una part del SCM que es formarà finalment.

En l'estat inicial, anomenat estat buit, es seleccionaran dos vèrtexs nuls.

Al llençar la funció $NextPair(n1, n2)$, per cada estat, es seleccionaran un parell de vèrtex que encara no s'ha analitzat, essent el primer vèrtex del primer graf i el segon vèrtex del segon graf.

El parell de vèrtexs seleccionats s'analitzen amb la funció $IsFeasiblePair(n1, n2)$ que comprova si és possible ampliar el subgraf comú amb l'estat actual, que representa l'esmentat parell de vèrtexs.

Llavors, si l'extensió és possible, la funció $AddPair(n1, n2)$ estèn la solució parcial amb el parell de vèrtex (n1,n2).

Immediatament després, si l'estat actual s no és una fulla de l'arbre de cerca, és còpia a si mateix mitjançant el llençament la funció $CloneState(s, s')$ i, a continuació, es comença l'anàlisi del nou estat.

Després de l'anàlisi del nou estat s'invoca una funció de retrocés (*backtracking*), restaura el subgraf comú de l'estat anterior i es tria un nou estat.

Si es fa ús d'aquesta estratègia de cerca, cada cop que es trii una branca, s'explorarà tan profundament com sigui possible fins que s'assoleixi una fulla.

Cal assenyalar que s'han de seguir totes les branques de l'arbre de cerca perquè, exceptuant els casos trivials, no és possible preveure si existeix una solució millor en una branca que encara no s'ha explorat.

S'ha de destacar també que, quan un estat no s'utilitzarà més s'esborrarà de la memòria al llençar la funció $Delete(s')$.

Sigui N_1i N_2 el nombre de vèrtexs del primer i segon graf, respectivament, en el pitjor dels casos el nombre d'estats que s'examinaran serà:

$$S = N_2! \left(\frac{1}{(N_2 - N_1)} + \ldots + \frac{1}{N_2 - 1} \right)$$

I, pel cas en què $N_1 = N_2 = N$ i $N >> 1$, l'anterior equació quedarà:

$$S = e \cdot N!$$

2.2.3 Un algorisme per trobar el SCM basat en la detecció de colles

L'algorisme de Durand-Pasari [8] es basa en la reducció de la cerca del SCM entre dos grafs al problema de trobar el CM d'un graf.

El primer pas és la construcció del graf associat, els vèrtexs del qual corresponen als vèrtexs dels dos grafs de partida amb la mateixa etiqueta i les arestes representes la compatibilitat del parell de vèrtexs de ser inclosos. Per la qual cosa, es pot obtenir el SCM fent la cerca del CM al graf associat.

L'algorisme de la detecció del CM genera una llista de vèrtexs que representa una colla del graf associat utilitzant una estratègia de cerca en profunditat (en anglès $depth - first\, search$) en l'arbre de cerca per seleccionar de manera sistemàtica un vèrtex a la vagada dels successius nivells, fins que no sigui possible afegir més vèrtexs a la llista. A continuació podem veure un esbós de l'algorisme per la detecció de colles maximals a 2.2.

Quan es considera un vèrtex es mira si és legal i, si ho és, l'algorisme comprova si la mida del nova colla és major que la mida de l'actual. Si la resposta és afirmativa es desa la nova colla.

En quan a la definició de legalitat, direm que un vèrtex és legal si està connectat a la totalitat de vèrtexs existents a la colla.

A cada nivell 'l', l'elecció dels vèrtexs a considerar és limitat als que corresponguin als parells (n_1, n_2) amb $n_1 = l$. D'aquesta manera l'algorisme assegura que l'espai de cerca és realment un arbre, és a dir, que no es

Algorithm 2.2 Esbós de l'algorisme per la detecció de CM

```
Procedure MCS_DP (vert_list)
Begin
    Level = lenght (vert_list)
    null_count = count_null_vertices (vert_list)
    clique_lenght = level-null_count;
    if(null_count >= best_null_count_so_far) then
        return;
    else if (level == max_level) then
        save(vert_list)
        best_null_count_so_far = null_count;
    else
        P = set of vertices (n1,n2) having n1 == level;
        Foreach (v in P ∪{NULL_VERTEX})
        Begin
            if(is_legal (v,vert_list)) then
                MCS_DP (vert_list + v);
            end if
        end
    end if
end procedure
```

tindrà en compte dues vegades la mateixa llista de vèrtexs.

Després de considerar tots els vèrtexs del nivell l, un vèrtex especial anomenat *vèrtex nul*, és afegit a la llista. L'esmentat vèrtex es considera sempre legal, pot ser afegit més d'un cop a la llista i s'utilitza per dur la informació que no hi ha cap assignació associada a un vèrtex en particular al primer graf, essent coincident.

Quan tots els vèrtexs possibles (inclòs el vèrtex nul) s'han considerat l'algorisme retrocedeix i tendeix a espandir-se per la llargària d'una branca diferent de l'arbre de cerca.

La longitud de la llista més llarga (amb l'exclusió de totes les entrades de vèrtexs nuls) és tan bona com la seva composició és mantinguda i aquesta informació és actualitzada segons sigui necessària.

Si N_1 i N_2 són el nombre de vèrtexs dels grafs inicials, amb $N_1 \leq N_2$, l'algorisme d'execució necessitarà un màxim de N_1 nivells i atès que a cada nivell l'espai que es necessita és constant, l'algorisme necessitarà un espai total de $\mathcal{O}(N_1)$.

En el pitjor dels casos el graf associat pot ser un graf complet de $N_1 \cdot N_2$ vèrtexs i l'algorisme haurà d'explorar $(N_2 + 1)$ vèrtexs al nivell 1, N_2 vèrtexs al nivell 2 fins als $(N_2 - N_1 + 2)$ vèrtexs al nivell N_1. Multiplicant aquests nombres obtenim que en el pitjor dels casos el nombre d'estats a explorar serà:

$$S = (N_2 + 1)(N_2) \ldots (N_2 - N_1 + 2) = \frac{(N_2 + 1)!}{(N_2 - N_1 + 1)!}$$

I, pel cas en què $N_1 = N_2 = N$, l'anterior equació es reduirà a:

$$\mathcal{O}(N \cdot N!).$$

2.2.4 Conclusions

En aquest article s'han descrit dos algorismes per trobar el SCM i el seu rendiment, en una base de dades que conté parells de grafs connectats aleatòriament, amb com a mínim un parell de vèrtexs.

Els tests comparatius preliminars mostren que, per una banda, per a grafs amb una densitat baixa, és a dir, amb un nombre de baix de vèrtexs, és més convenient cercar el SCM mitjançant el primer algorisme definit. Mentre que, per altra banda, quan els dos grafs donats tenen una densitat alta és més convenient construir el graf associat per a cercar el CM.

2.3 El problema del SCM

2.3.1 Introducció

El problema de trobar el subgraf comú maximal (SCM) és ben conegut i consisteix en, donats un parell de grafs, cercar el subgraf comú maximal induït d'ambdós. Actualment, existeixen una gran quantitat d'algorismes per trobar l'òptim però el temps d'execució que empren els esmentats són similars als temps necessitats en un atac per força bruta, que és exponencial.

2.3.2 Solucions exactes pel problema del SCM

En el procés de cerca del subgraf comú, es tractar de trobar una funció f un a un, de $V(G_1)$ a $V(G_2)$, de tal manera que un subconjunt del domini i la seva imatge siguin subgrafs induïts isomorfs de G_1 i G_2, respectivament. En aquest context, si $v = f(u)$, direm que u i v coincideixen o que u i v s'aparellen.

Sigui $M = S_1 \times S_2 \subset V(G_1) \times V(G_2)$ tal que S_1 i S_2 indueixen subgrafs isomorfs. Ens referim a M com un conjunt de parelles compatibles. Si un parell (u, v) de $(V(G_1) - S_1) \times (V(G_2) - S_2)$ de manera que $G_1[S_1 \cup \{u\}]$ és isomorf a $G_2[S_2 \cup \{v\}]$, llavors diem que el parell (u, v) és compatible amb M.

Algorismes de força bruta i backtracking

Per trobar un subgraf comú maximal, les tècniques per força bruta, per definició, han d'explorar l'espai de tots els possibles subgrafs comuns maximals.

Una manera de fer-ho és enumerar totes les funcions de $V(G_1)$ a $V(G_2) \cup \{NONE\}$, on $NONE$ és un node fictici que serveix com a imatge dels vèrtexs no aparellats de G_1.

Llavors, el nombre de totes les funcions és $(m + 1)n$, però no totes elles són isomorfismes entre subgrafs de G_1 a G_2. De fet, llevat que es tracti d'alguns cassos especials (com el cas trivial) la majoria d'aquestes funcions no són importats per nosaltres (en el sentit que no són subgrafs isomorfs vàlids).

Per tant, $(m + 1)n$ és un límit superior baix. De totes les funcions de $V(G_1)$ a $V(G_2)$ cerquem una que satisfà les condicions següents:

1. La restricció de f a $D_f = V(G_1) - \{v \in V(G_1) : f(v) = NONE\}$ és una funció un a un.
2. Els subgrafs induïts per D_f i $f(D_f)$ són isomorfs.

Durant la cerca del subgraf comú maximal s'ha de fer un seguiment dels elements següents:

- El major conjunt, M, de parells compatibles trobats.
- El cardinal d'M, denotat per maxsize.
- Per cada $i \in G_1$: el conjunt $M_1[i]$ de tots els vèrtexs de G_2 que podrien ser aparellats amb i.
- Per cada $i \in G_2$: el conjunt $M_2[i]$ de tots els vèrtexs de G_1 que podrien ser aparellats amb i.
- Per cada $i \in G_j$: el conjunt $M_j[i]$ denotat per $matchnum_j[i]$ $j \in (1, 2)$.

Llavors, es farà una cerca per branques com si s'explorés un arbre de cerca T. Cada node de T té un conjunt actual d'parells compatibles, anomenat $currentM$, que consisteix en tots els parells que han anat a l'una, per tant considerada compatible fins al moment (al llarg del ruta des de l'arrel al node actual). La mida del $currentM$ s'anomena $currentsize$.

S'ha de tenir en compte que T és un arbre virtual. És explorat (mitjançant $depth - first$), però no es construeix de forma explícita. Cada node de T es podria visualitzar com un etiquetatge amb un vèrtex de G_1 que és seleccionat segons un criteri heurístic (com el valor mínim de $matchnum1$).

Les arestes de T són etiquetades amb vèrtexs de G_2 o $NONE$. S'ha de tenir en compte que dos o més vèrtexs de T poden tenir la mateixa etiqueta, llevat que pertanyin a la mateixa ruta des de l'arrel a una fulla (de T).

Si el vèrtex actual està vinculat al seu pare u a través de l'aresta x, quan $u \in G_1$ s'ha emparellat amb $x \in G_2$ al pas anterior. A més, en aquest cas, (u, x) és un element del conjunt $currentM$ trobat en el camí des de l'arrel fins al fill del node u que està connectat amb u a través de l'aresta x.

Si no s'han emparellat al pas (o nivell) anterior, llavors l'etiqueta de l'aresta serà $NONE$ (en lloc d'x).

Sigui v el node actual en el procés de cerca, llavors v ($\in V(G_1)$) es podria associar amb un vèrtex, per exemple y, a partir de $M_1[v] \cup NONE$.

Per a cada associació de v per a un vèrtex de $M_1[v]$, es seguiran els passos següents:

1. Esborrar y de $M_1[i], \forall i \in V(G_1)$;
2. Esborrar v de $M_2[j], \forall j \in V(G_2)$;
3. Afegir (v, y) a $currentM$ i incrementar $currentsize$;
4. Si $currentsize > maxsize$, llavors $M \leftarrow currentM$;
5. A continuació seleccionem un altre vèrtex de G_1.

Desfer els passos anteriors és un dels colls d'ampolla de la implementació. En particular, el codi ha de donar compte de la supressió temporal dels vèrtexs de M_1 i M_2.

La implementació es pot fer de moltes maneres, motiu pel qual en el present article no es detallen. El que ha de quedar clar és que, actualment, un algorisme recursiu de *backtracking* per trobar el SCM té un cost temporal elevat, en grafs de mides petites.

L'anterior proposta d'algorisme de *backtracking* és similar a la proposada de [9], que actualment ofereix la millora més gran del vell algoritme de *backtracking* proposat per Ullman en la seva obra sobre isomorfismes de subgrafs [10].

Com s'ha dit anteriorment, els algoritmes de *backtracking* pateixen del seu comportament pitjor dels casos, el que sembla inevitable. El pitjor dels casos en temps d'execució de l'algorisme anterior és $\mathcal{O}((m+1)^n)$.

L'ús del grafs compatibles

Un altre enfocament al problema de trobar el SCM és cercar el conjunt més gran de parells compatibles en el graf associat, G_A. És a dir, per obtenir el subgraf comú maximal es necessita el conjunt més gran de parells compatibles, que ve a ser la colla maximal de $G_{A(G_1,G_2)}$.

I, l'algorisme més conegut per trobar la colla maximal té un cost temporal, en el pitjor dels cassos de l'ordre de $\mathcal{O}(2^{|A(G_1,G_2)|/4})$, que és $\mathcal{O}(2^{mn/4})$ [11].

D'un cost d'execució, que és pitjor que el mètode d'atac mitjançant força bruta descrit anteriorment. Tot i que els darrers anàlisis han demostrat que els mètodes basats amb colles poden prendre avantatges pel que

fa a l'estructura, són relativament fàcils d'implementar i tenen el mateix comportament que en els mètodes de backtracking en força bruta en el pitjor dels casos [12].

Aquests resultats es basen en explorar el fet que les parelles amb un component comú no podran ser part de la colla desitjada, una propietat que és assumida en els mètodes de backtracking perquè els vèrtexs trobats són posteriorment eliminats o ignorats.

2.3.3 Conclusions

Com s'ha pogut veure en aquest article, trobar el subgraf comú maximal és un problema de cost exponencial. Cal remarcar que hi pot haver una solució més ràpida per tal de trobar el SCM, que consisteix en transformar una part del procés de cerca en la tasca d'enumerar el conjunts maximals independents en només una part de cadascun dels grafs introduïts, que s'anomena cobertura de vèrtexs (*vertexcover*).

Capítol 3

Llibreria NetworkX

En aquest capítol hi ha una breu introducció a la llibreria *NetworkX*, on es poden veure algunes de les funcions que conté i que s'han utilitzat per la implementació dels algorismes que han estat objecte de l'estudi i treball.

3.1 Introducció a la llibreria NetworkX

NetworkX[1] és una llibreria feta amb *Python* per l'estudi de xarxes i grafs. La llibreria és de programari lliure[2] i distribuïda sota la nova llicència de *BSD*. I, actualment, és integrada dins del projecte *SAGE*[3].

Les seves principals característiques són:

- Conté classes per grafs i digrafs.
- Hi ha la possibilitat de conversió de grafs cap a/des de diversos formats.
- Permet la construcció aleatòria de grafs i/o de forma creixent.
- Té funcions per cercar i trobar subgrafs, colles, etc.
- Té funcions per cercar i calcular adjacències, graus, diàmetres, centres, etc.
- Dibuixa xarxes en 2D i 3D.
- Té més de 1.000 proves unitàries.
- Permet un fàcil accés a la majoria de bases de dades.
- És multiplataforma[4].

Per tant, *NetworkX* és molt adequada per fer operacions amb mides grans de grafs en el món real, com, per exemple, grafs de 10 milions de vèrtexs i 100 milions d'arestes, gràcies a l'estructura de dades *diccionari de diccionaris*, pròpia de *Python*.

En resum, *NetworkX* és ranablement eficient, molt escalable i altament portable en el marc de l'anàlisi de xarxes.

[1] http://networkx.lanl.gov/

[2] El programari lliure (en anglès *free software*) és el programari que es pot utilitzar, estudiar i/o modificar sense restriccions, i que pot ser copiat i redistribuït. en una versió modificada o sense modificar, sense cap restricció o amb unes restriccions mínimes, per garantir que els futurs destinataris també tindran aquests drets.

[3] http://sagemath.org/links-components.html

[4] Multiplataforma és un terme informàtic que s'utilitza per definir al programari, ja sigui un sistema operatiu, llenguatge de programació, programa, etc. que pot ésser executat en diverses plataformes.

3.2 Funcions per colles

3.2.1 Funció `find_cliques`

Mètode: `networkx.algorithms.clique.find_cliques (G)`
Crida: `find_cliques (G)`
Explicació: cerca les colles maximals d'un graf G. La sortida genera una llista de llistes, i cadascuna de les darreres és una colla maximal formada per un reguitzell de vèrtexs.

3.2.2 Funció `make_max_clique_graph`

Mètode: `networkx.algorithms.clique.make_max_clique_graph (G,create_using=None,name=None)`
Crida: `make_max_clique_graph (G, create_using=None, name=None)`
Explicació: crea un graf de colles maximals. Troba les colles maximals i les tracta com si fossin vèrtexs. Els vèrtexs estaran connectats si els seus membres són comuns en el graf original.

3.2.3 Funció `make_clique_bipartite`

Mètode: `networkx.algorithms.clique.make_clique_bipartite (G,fpos=None, create_using=None, name=None)`
Crida: `make_clique_bipartite (G, fpos=None, create_using=None, name=None)`
Explicació: crea un graf bipartit de colles, B, d'un graf donat, G. Els vèrtexs inferiors de B són els vèrtexs originaris de G i els vèrtexs superiors de B són les colles que hi ha en G. Hi haurà una aresta si un vèrtex inferior pertany a la colla representada pel seu vèrtex superior.

3.2.4 Funció `graph_clique_number`

Mètode: `networkx.algorithms.clique.graph_clique_number (G,cliques=None)`
Crida: `graph_clique_number (G, cliques=None)`
Explicació: retorna el cardinal de la colla maximal d'un graf G, és a dir, el nombre de vèrtexs que la componen. Dóna l'opció de ficar una llista de colles si ja està calculada.

3.2.5 Funció `graph_number_of_cliques`

Mètode: `networkx.algorithms.clique.graph_number_of_cliques (G,cliques=None)`
Crida: `graph_number_of_cliques (G, cliques=None)`
Explicació: retorna el nombre de colles maximals del graf G passat com a argument, és a dir, el seu cardinal. Dóna l'opció de ficar una llista de colles si ja està calculada.

3.2.6 Funció `node_clique_number`

Mètode: `networkx.algorithms.clique.node_clique_number (G,nodes=None,cliques=None)`
Crida: `node_clique_number (G, nodes=None, cliques=None)`
Explicació: entrat un vèrtex, d'un graf G, retorna el nombre (cardinal) de la colla maximal en què està inclòs. Si s'ha entrat més d'un vèrtex retorna una llista. Dóna l'opció de ficar una llista de colles si ja està calculada.

3.2.7 Funció `number_of_cliques`

Mètode: `networkx.algorithms.clique.number_of_cliques (G,nodes=None,cliques=None)`
Crida: `number_of_cliques (G, nodes=None, cliques=None)`
Explicació: retorna el nombre (cardinal) de colles maximals per cada vèrtex del graf G. Dóna l'opció de ficar una llista de colles si ja està calculada.

3.2.8 Funció `cliques_containing_node`

Mètode: `networkx.algorithms.clique.cliques_containing_node (G,nodes=None,cliques=None)`
Crida: `cliques_containing_node (G, nodes=None, cliques=None)`
Explicació: retorna una llista de colles pel vèrtex del grag G passat com argument. Dóna l'opció de ficar una llista de colles si ja està calculada.

3.3 Altres funcions

3.3.1 Funció `read_adjlist`

Mètode: `networkx.readwrite.adjlist.read_adjlist (path, comments='#', delimiter=' ', creat e_using=None, nodetype=None, encoding='utf-8')`
Crida: `adjlist.read_adjlist (path, comments='#', delimiter=' ', create_using=None, nodetyp e=None, encoding='utf-8')`
Explicació: llegeix el graf desat en format de llista d'adjacències, d'una adreça donada, i retorna el graf corresponent. Dóna l'opció de posar comentaris, de definir els separadors de les etiquetes dels vèrtexs i convertir-los al tipus de *Python* triat.

3.3.2 Funció `write_adjlist`

Mètode: `networkx.readwrite.adjlist.write_adjlist (G, path, comments='#', delimiter=' ', e ncoding='utf-8')`
Crida: `write_adjlist (G, path, comments='#', delimiter=' ', encoding='utf-8')`
Explicació: escriu el graf G en format de llista d'adjacències, en l'adreça passada com a paràmetre. Dóna l'opció de posar comentaris i de definir els separadors de les etiquetes dels vèrtexs.

3.3.3 Funció `order`

Mètode: `networkx.DiGraph.order()`
Crida: `order()`
Explicació: retorna el nombre de nodes del graf que fa la crida del mètode.

3.3.4 Funció `add_nodes_from`

Mètode: `networkx.DiGraph.add_nodes_from (nodes, **attr)`
Crida: `add_nodes_from (nodes, **attr)`
Explicació: afegeix múltiples nodes al graf que fa la crida del mètode. És un contenidor de nodes on es poden posar atributs, ja sigui una llista, diccionari, etc.

3.3.5 Funció `has_edge`

Mètode: `networkx.DiGraph.has_edge (node u, node v)`
Crida: `has_edge (node u, node v)`
Explicació: retorna un booleà que s'avalua a cert si l'aresta (u, v) és al graf que fa la crida del mètode, i fals altrament.

3.3.6 Funció `add_node`

Mètode: `networkx.DiGraph.add_edge (node u, node v, attr_dict=None, **attr)`
Crida: `add_edge (node u, node v, attr_dict=None, **attr)`
Explicació: afegeix una aresta entre els vèrtexs u i v. Opcionalment es pot posar un diccinari d'atributs de les arestes.

3.3.7 Funció `random_regular_graph`

Mètode: `networkx.generators.random_graphs.random_regular_graph(int d, int n, create_using =None, seed=None)`
Crida: `random_regular_graph(d, n, create_using=None, seed=None)`
Explicació: retorna un graf d'n vèrtexs amb el mateix grau d, que és creat de forma aleatòria. El nombre de vèrtexs n que se li passa com a paràmetre ha de ser parell. Opcionalment se li pot passar un graf com a instància i una llavor per generar els nombres aleatoris. Els vèrtexs seran numerats des de 0 fins a $(n-1)$.

Capítol 4

Implementació dels algorismes

En aquest capítol es mostren i expliquen les implementacions dels algorismes que han estat necessaris per a l'assoliment de l'estudi del treball.

4.1 Implementació del graf associat

4.1.1 Explicació

El primer que fem és crear els grafs buits $g1$, $g2$ i $gass$ per tal de poder manipular la informació. A continuació recuperem les llistes d'adjacències dels grafs $g1$ i $g2$, que prèviament haurem d'haver desat a l'ordinador (en el cas de l'exemple, les adreces són `'/home/david/TFC/Memòria fitxers/g1-generat.lla'`, pel graf $g1$, i `'/home/david/TFC/Memòria fitxers/g2-generat.lla', pel graf` $g2$`).`

Posteriorment calculem l'ordre del graf associat, per fer-ho multipliquem els ordres dels grafs obtinguts per les llistes d'adjacències, tal i com diu la definició de graf associat. I per obtenir la llista d'arestes ho farem tenint en compte que, si no es modifica expressament, NetworkX anomena els vèrtexs d'un graf d'ordre n de la següent forma: $0, 1, 2, ..., n-1$, de tal manera que en el graf associat els vèrtexs s'etiquetarien com parells ordenats de 0 fins a n_1 i de 0 fins a n_2, essent n_1 i n_2 l'ordre dels grafs $g1$ i $g2$ respectivament.

Així que compararem els parells ordenats amb els seus respectius grafs i si compleixen les condicions assenyalades en la definició afegirem una aresta al graf associat, anomenat $gass$, tenint en compte que els vèrtexs es calculen en funció al seu ordre.

4.1.2 Codi font

A l'algorisme 4.1 es mostra el codi font del graf associat amb *Python*.

4.2 Implementació del càlcul de la distància

4.2.1 Explicació

Per fer el càlcul de la distància ens basarem en la definició 7 de l'apartat 2.1.3. On $g1$ i $g2$ seran dos grafs qualsevols i $gass$ el seu graf associat. A la unitat li restarem el quocient que surt de calcular la mida del clique maximal de $gass$, dividit per l'ordre màxim de comprarar l'ordre de $g1$ amb l'ordre de $g2$.

Llavors, per calcular la mida del clique maximal de $gass$, hi ha les opcions següents:

- Una primera opció és fer ús de la funció `graph_clique_number` 3.2.4. Tal i com podem veure en l'algorisme 4.2.
- I, una segona opció que és recórrer cadascun dels nodes del graf fent ús de la funció `node_clique_number` 3.2.6 (es podria implementar, que a la vegada, es comparés el resultat calculat amb l'ordre del graf, ja que si aquest fos idèntic es podria aturar l'execució). Tal i com podem veure en l'algorisme 4.3.

Com que la primera és computacionalment més eficient i d'un cost temporal més baix, és la que s'ha emprat finalment.

4.2.2 Codi font

Als algorismes 4.2 i 4.3 es mostra el codi font amb *Python* de les dues opcions que es poden utilitzar per cercar la colla maximal i així fer el posterior càlcul per obtenir la distància entre els grafs desitjada.

Algorithm 4.1 Codi font del graf associat

```
#!/usr/bin/env python
# -*- coding:  utf-8

import sys
from networkx import *

def calcul_graf_associat(g1,g2):
    gass=Graph()
    ordre_gass=g1.order()*g2.order();
    gass.add_nodes_from(range(ordre_gass))

    for k in range(g1.order()):
        for l in range(g2.order()):
            for m in range(g1.order()):
                for n in range(g2.order()):
                    if (k!=m and l!=n and
                    ((g1.has_edge(k,m) and g2.has_edge(l,n)) or
                    ((not(g1.has_edge(k,m))) and (not(g2.has_edge(l,n))))):
                        a=k*g2.order()+l
                        b=m*g2.order()+n
                        gass.add_edge(a,b)
    write_adjlist(gass,"/home/david/TFC/Memòria fitxers/gass-generat.lla");
    return gass
```

Algorithm 4.2 Codi font de la distància amb graph_clique_number

```
#!/usr/bin/env python
# -*- coding:  utf-8

import sys
from networkx import *

def calcul_distancia(gass,g1,g2):

    mcs_max=graph_clique_number(gass,None)

    max_g1_g2=max(g1.order(),g2.order())
    d= 1-(float(mcs)/float(max_g1_g2))
    return d
```

Algorithm 4.3 Codi font de la distància amb node_clique_number

```
#!/usr/bin/env python
# -*- coding:  utf-8

import sys
from networkx import *

def calcul_distancia(gass,g1,g2):
    mcs=0
    for i in range(gass.order()):
        if (mcs<node_clique_number(gass,i,None)):
            mcs=node_clique_number(gass,i,None)

    max_g1_g2=max(g1.order(),g2.order())
    d= 1-(float(mcs)/float(max_g1_g2))
    return d
```

Capítol 5

Avaluació i dades

En aquest capítol es fa una anàlisi de les dades extretes d'avaluar els programes realitzats pel càlcul de la distància entre grafs en un ordinador anomenat *turing.udl.cat*, les especificacions del qual es detallen a la Subsecció 5.3.

En concret s'han avaluat les distàncies que hi ha entre el graf de Petersen i la família de grafs radials de Moore de diàmetre 3 i radi 2, que es poden veure en el següent apartat. I el cost temporal que suposa avaluar grafs 3-regular amb diferents nombres de vèrtexs, que és el motiu pel qual no s'ha pogut calcular la distància pels graf de Moore de diàmetre i radi igual a 3.

5.1 Càlculs entre el graf de Petersen i els grafs de la família radials de Moore

5.1.1 Distàncies i temps de càlcul

En la taula taula següent es poden veure els resultats d'executar els programes per calcular la distàncies entre el graf de Petersen i la família de grafs radials de Moore de diàmetre 3 i radi 2. És a dir, es mostra la distància que hi ha entre cadascun dels grafs si es pren com a referència el graf de Petersen. També es mostra el temps de còmput necessitat per tal de portar a terme els càlculs.

Nom del graf	Vèrtexs	Arestes	Distància	Temps (s)
Petersen	10	15	0.0	0,0550437570000
iso.2.1	10	15	0.3	0.1191158294680
iso.2.2	10	15	0.3	0.0941491127014
iso.2.3	10	15	0.2	0.0931360721588
iso.2.4	10	15	0.2	0.0742161273956

Taula 5.1: Distàncies i temps per la família de radials de Moore de diàmetre 3 i radi 2

Com es pot observar, la distància del graf de Petersen amb ell mateix sempre serà zero per definició, un graf és isomorf amb ell mateix. I la resta de distàncies s'han calculat utilitzant la definició número 7 de l'apartat 2.1.3.

5.1.2 Exemple concret

A les Figures 5.1 i 5.2 podem visualitzar els grafs de *Petersen* i el graf *radial de Moore iso*.2.4 de diàmetre 3 i radi 2, i el seu SCM, respectivament.

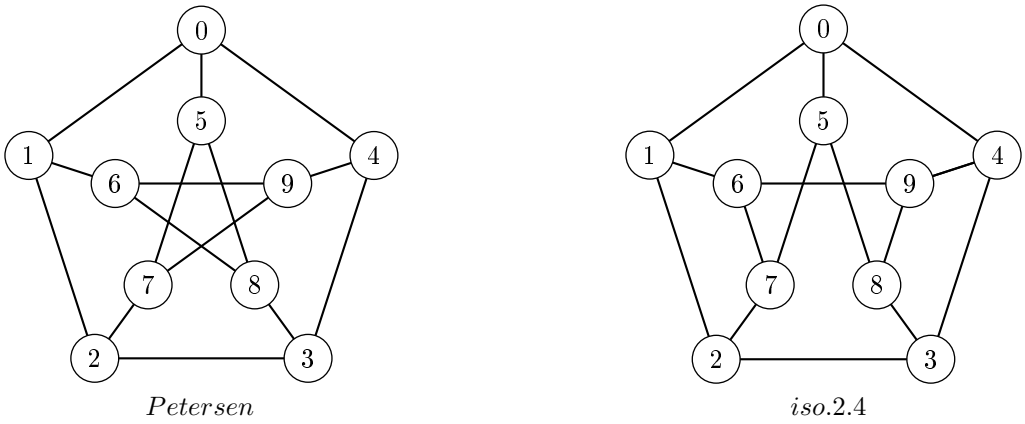

Figura 5.1: Grafs de Petersen i radial de Moore 2.4

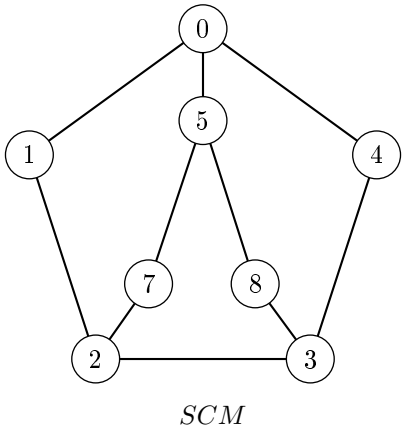

Figura 5.2: SCM dels grafs de Petersen i radial de Moore 2.4

5.2 Altres càlculs

5.2.1 Distàncies i temps de càlcul per grafs 3-regular amb diferent nombre de vèrtexs

Inicialment els càlculs de les distàncies pel treball de final de carrera s'anaven a avaluar per la família de grafs radials de Moore de diàmetre 3 i radi igual a 2 i per la família de grafs de Moore de distància i radi igual a 3.

Les primeres distàncies s'han pogut calcular amb èxit ja que les operacions exigides no han suposat un gran esforç en temps d'execució. Però les distàncies per la segona família no s'han pogut portar a terme ja que el temps necessitat per avaluar els 1.062 grafs que formen la família suposa un cost temporal massa elevat per la potència de l'ordinador en què s'han dut a terme les càlculs.

Si, es suposa i es pren com a referència les dades de la taula 5.2 podem observar que per grafs 3-regular d'ordre 20 tenim que el temps emprat és aproximadament de 5 h i 30 min, i sabem que la funció per calcular colles maximals té un creixement exponencial, per fer el càlcul del SCM d'ordre 22 podria trigar un mínim aproximat de 2 dies i mig, el que es traduiria amb un temps total de còmput de 7 anys i mig per tota la família.

Nom del graf	Vèrtexs	Arestes	Distància	Temps (s)
Graf 1	12	18	0.25	0.953146934509
Graf 2	14	21	0.214285714286	9.650382995610
Graf 3	16	24	0.1875	107.6162061690
Graf 4	18	27	0.222222222222	1459.920843120
Graf 5	20	30	0.2	19718.01788000

Taula 5.2: Distàncies entre grafs 3-regulars aleatoris amb el mateix número de vèrtexs

El nombre d'arestes de la taula ha estat calculat *pel lema de les encaixades de mans*. És a dir, pel teorema on es relacionen els graus dels vèrtexs d'un graf, on s'afirma que la suma de tots els graus dels vèrtexs d'un graf G=(V,A) és igual al doble de la mida de G, és a dir,

$$\sum_{v \in V} g(v) = 2|A| \tag{5.1}$$

i, al ser un graf d-regular, es dedueix:

$$d|V| = 2|A| \quad \Rightarrow \quad |A| = \frac{d|V|}{2}$$

5.3 Especificacions de "turing.udl.cat"

Les especificacions de la màquina en què s'han realitzat els càlculs són les següents:

- Nom: turing.udl.cat
- Propietari: Universitat de Lleida (UdL)
- Sistema operatiu: Linux
- CPU (Unitat de Procés Central):
  - 2 processadors físics Intel(R) Xeon(R) CPU X5460 @ 3.16GHz distribuïts amb 8 nuclis diferents, versió 6.7.6
  - mida: 3163MHz, capacitat: 3166MHz, ample: 64 bits i rellotge: 1333MHz
  - memòria cau L1 amb mida: 128KiB i capacitat: 128KiB
  - memòria cau L2 amb mida: 12MiB i capacitat: 12MiB
- Memòria: mida de 16GiB distribuïda en 8 ranures (*slots*) amb l'especificació següent per cadascuna:
  - Descripció: Synchronous 667 MHz (1.5 ns)
  - Mida: 2GiB
  - Ample: 64 bits
  - Rellotge: 667MHz (1.5ns)

5.4 Relació de fitxers

Els fitxers que hi ha en el CD adjunt en la memòria són:

- "llibreriaGrafAssociat.py": calcula el graf associat de dos grafs donats.
- "llibreriaDistancia2.py": calcula la distància empran la funció `graph_clique_number`.
- "programa_projecte_regulars.py": calcula els costos temporals de grafs 3-regular amb diferent nombre de vèrtexs.
- "programa_projecte_rMd3r2.py": calcula les distàncies i costos temporals entre el graf de Petersen i la família de radials de Moore amb diàmetre 3 i radi 2, la família la llegeix recursivament del directori rMd3r2.
- "programa_projecte_rMd3r3.py"": calcula les distàncies i costos temporals entre el graf de Petersen i la família de radials de Moore amb diàmetre 3 i radi 3, la família la llegeix recursivament del directori rMd3r3.

Per fer servir els programes només fa falta posar-los en mode execució i tenir una carpeta "distancies" creada i les dues carpetes que contenen les famílies de grafs radials de Moore esmentades.

Capítol 6

Resum, conclusions i possible treball futur

Pel tal d'assolir els coneixements necessaris per a la comprensió i elaboració del projecte, s'han hagut de repassar conceptes teòrics de la teoria de grafs, incidint en els conceptes d'isomorfisme, distància de grafs, graf associat, subgraf comú maximal i colles maximals. A més, s'ha hagut que treballar la bibliografia relativa als articles dels conceptes esmentats, posant especial èmfasi en els articles [1, 2, 3].

També s'han hagut d'adquirir les nocions bàsiques pel que fa al llenguatge de programació *Python* [13, 14] per tal de manipular les dades requerides per fer ús de les funcions implementades en la llibreria *NetworkX* [15] i la creació de les funcions necessàries per la consecució de la tasca que s'ha dut a terme.

A partir d'aquest treball previ s'ha aconseguit dissenyar i implementar un programa que calcula, donats dos grafs qualssevol, el seu graf associat, per tal de poder cercar la colla maximal, del darrer. A més s'ha comprovat el seu funcionament sobre la família de grafs radials de Moore de diàmetre 3 i radi 2 i s'han avaluat els costos temporals aproximats per a la família de radials de Moore amb diàmetre i radi igual a 3.

Com a proposta de treball futur immediat es pot calcular la distància de la família de grafs de Moore de diàmetre i radi 3, en un ordinador amb la capacitat suficient per realitzar els càlculs en un termini de temps raonable. Estudiar algorismes més eficients per a la resolució del problema. O bé, valorar la possibilitat de calcular les distàncies per digrafs, modificant la funció de graf associat, entre d'altres.

Bibliografia

[1] Horst Bunke[1] i Kim Shearer[2], "A graph distance metric based on the maximal common subgraph", [1]Institut fur Informatik und angewandte Mathematik, University of Bern, Bern, Switzerland i [2]Department of Computer Science, Curtin University of Technology, Perth, WA, Australia.

[2] Horst Bunke, Pasquale Foggia, Corrado Guidobaldi, Carlo Sansone i Mario Vento, "A Comparison of Algorithms for Maximum Common Subgraph on Randomly Connected Graphs", IEEE in Proc. SSPR/SPR, 2002, pp.123-132.

[3] Faisal N. Abu-Khzam, Nagiza F. Samatova, Mohamad A. Rizk i Michael A. Langston, "The Maximum Common Subgraph Problem: Faster Solutions via Vertex Cover", IEEE in Proc. AICCSA, 2007, pp.367-373.

[4] Joan Gimbert, Ramiro Moreno, Josep Maria Ribó i Magda Valls, "Apropament a la teoria de grafs i els seus algorismes". Edicions de la Universitat de Lleida, 1998. ISBN: 84-89727-65-1.

[5] Levi, G., 1972. "A note on the derivation of maximal common subgraphs of two directed or undirected graphs". Calcols 9, 341–354.

[6] J. J. MacGregor, "Backtrack Search Algorithms and the Maximal Common Subraph Problem", Software Practice and Experience, Vol. 12, pp. 23-34, 1982.

[7] Bunke, H., 1997. "On a relation between graph edit distance and maximum common subgraph". Pattern Recognition Lett. 18 Ž8., 689–694.

[8] P. J. Durand, R. Paasari, J. W. Baker, and Chun-che Tsai, "An Efficient Algorithm for Similary Analysis of Molecules", Internet of Chemistry, vol. 2, 1999.

[9] E. B. Krissinel and K. Henrick. "Common subgraph isomorphism detection by backtracking search". Software Practice and Experience, 34:591–607, 2004.

[10] J. R. Ullman. "An algorithm for subgraph isomorphism". Journal of the ACM, 23(1): 31–42, 1976.

[11] J. M. Robson. "Finding a maximum independent set in time $\mathcal{O}(2n/4)$". Technical Report 1251-01, Universite Bordeaux I, LaBRI, 2001.

[12] W. Henry Suters, F. N. Abu-Khzam, Y. Zhang, C. T. Symons, N. F. Samatova, and M. A. Langston. "A new approach and faster exact methods for the maximum common subgraph problem". In 11th International Computing and Combinatorics Conference, Kunming, China, Lecture Notes in Computer Science, volume 3595, pages 717–727. Springer, August 2005.

[13] "Python para todos". Raúl González Duque. `http://mundogeek.net/tutorial-python/`

[14] Documentació de Python `http://docs.python.org/index.html`

[15] NetworkX `http://networkx.lanl.gov/index.html`